EXAMINER'S AMENDMENT

An examiner's amendment to the record appears below. Should the changes and/or additions be unacceptable to applicant, an amendment may be filed as provided by 37 CFR 1.312. To ensure consideration of such an amendment, it MUST be submitted no later than the payment of the issue fee.

Authorization for this examiner's amendment was given in a telephone interview with Crystal Sayles on 10/21/10.

The application has been amended as follows:

1. A <u>computer implemented</u> code generation method, comprising:

a compiler generating a table of patterns, each pattern in the table comprising an FMA (fused multiply-add) DAG (Directed Acyclic Graph), a canonical form equivalent of the FMA DAG, and a shape corresponding to the canonical form equivalent; and

matching <u>a DAG representing</u> incoming floating point expressions <u>in a source code program received by the compiler</u> against the patterns in the table of patterns during compilation of a̶ <u>the</u> program<u>;</u>

<u>the compiler generating a sequence of optimized FMA instructions to compute the floating-point expressions</u>.

2. The method of claim 1, wherein generating the table of patterns occurs once during compilation of a̶ <u>the</u> compiler.

4. The method of claim 3 wherein arguments for each instruction in the sequence of

FMA instructions comprise <u>a number of</u> terminals a, b, c, ... and constants one (1) and

zero (0), wherein each terminal appears once in the sequence of FMA instructions and

the FMA DAG includes at least one node.

7. The method of claim 1, wherein generating a table of patterns comprises:

    generating all possible FMA DAGs of a predefined complexity or less;

    determining canonical forms and shapes for each FMA DAG;

    sorting the generated FMA DAGs according to shape;

    pruning the generated FMA DAGs;

    sorting each group of shapes according to complexity and height;

    encoding each <u>FMA DAG as a</u> pattern ~~into~~ <u>comprising</u> a 64-bit number; and

    storing the patterns as a table in a file.

15. The method of claim 13, wherein if the mapping is valid, the method further

~~comprising~~ <u>comprises</u> providing an ~~optimal~~ <u>optimized</u> sequence of FMA (fused multiply-

add), FMS (fused multiply-subtract), and/or FNMA (fused negate multiply-add)

instructions as compiled code for computing the incoming expression.

17. An article comprising:

a <u>non-transitory</u> storage medium having a plurality of machine accessible

instructions, wherein when the instructions are executed by a processor, the instructions

provide for generating a table of patterns, each pattern in the table comprising an FMA

(fused multiply-add) DAG (Directed Acyclic Graph), a canonical form equivalent of the

FMA DAG, and a shape corresponding to the canonical form equivalent; ~~and~~

matching <u>a DAG representing</u> incoming floating point expressions <u>in a source</u>

<u>code program received by a compiler</u> against the patterns in the table of patterns during

compilation of a program<u>; and</u>

<u>generating a sequence of optimized FMA instructions to compute the floating-</u>

<u>point expressions</u>.

18. The article of claim 17, wherein generating the table of patterns occurs once during

compilation of ~~a~~ <u>the</u> compiler.

20. The article of claim 19, wherein arguments for each instruction in the sequence of

FMA instructions comprise <u>a number of</u> terminals ~~a, b, c, ...~~ and constants one (1) and

zero (0), wherein each terminal appears once in the sequence of FMA instructions and

the FMA DAG includes at least one node.

23. The article of claim 17, wherein instructions for generating a table of patterns

comprises instructions for:

generating all possible FMA DAGs of a predefined complexity or less;

determining canonical forms and shapes for each FMA DAG;

sorting the generated FMA DAGs according to shape; pruning the generated

FMA DAGs;

sorting each group of shapes according to complexity and height;

encoding each <u>FMA DAG as a</u> pattern ~~into~~ <u>comprising</u> a 64-bit number; and

storing the patterns as a table in a file.

31. The article of claim 29, wherein if the mapping is valid, the ~~method~~ <u>instructions</u>

further comprising instructions for providing an ~~optimal~~ <u>optimized</u> sequence of FMA

(fused multiply-add), FMS (fused multiply-subtract), and/or FNMA (fused negate

multiply-add) instructions as compiled code for computing the incoming expression.

33: A code generation system, comprising:

a processor having an instructions set comprising fused instructions;

a memory, the memory comprising a code generator having a floating-point

module coupled to an optimizer and a table of patterns coupled to the optimizer,

<u>each pattern in the table comprising one of a FMA (fused multiply-add), FMS</u>

<u>(fused multiply-subtract), or FNMA (fused negate multiply-add) instruction DAG</u>

<u>(Directed Acyclic Graph), a canonical form equivalent of the FMA, FMS or FNMA DAG,</u>

<u>and a shape corresponding to the canonical form equivalent;</u>

~~the processor for enabling~~ the code generator <u>configured</u> to receive floating-point

expressions and to generate a sequence of ~~optimal fused multiply-add~~ <u>optimized FMA,</u>

~~fused multiply-subtract~~FMS, and/or ~~fused negate multiply-add~~FNMA instructions to

compute the floating-point instruction.

34: The system of claim 33, wherein ~~the processor to enable~~ the floating-point module is

configured to receive as input source code and to extract floating-point expressions

from the source code.

35: The system of claim 33, wherein ~~the processor to enable~~ the optimizer is configured

to receive the floating-point expression from the floating-point module and to determine

a canonical form and shape for the input floating-point expression.

36: The system of claim 35, wherein ~~the processor to further enable~~ the optimizer is

configured to search the table of patterns to find a pattern having a canonical form,

shape, and at least an equivalent amount of terminals in the DAG to that of the

canonical form, shape, and terminals of the input floating-point expression.

37: The system of claim 36, wherein ~~the processor to further enable~~ the optimizer is

configured to determine whether a valid mapping exists between the terminals of the

pattern and the terminals of a DAG representing the input floating-point expression, and

if there is a valid mapping, ~~the processor to further enable~~ the optimizer is further

configured to replace the terminals in the corresponding ~~FMA~~ DAG with the terminals

from the DAG representing the input floating-point expression and to determine sign

combinations to find a correct sign combination and canonical form of the DAG equal to

the incoming expression.

38. The system of claim 37, wherein ~~the processor to further enable~~ the optimizer <u>is</u>

<u>further configured</u> to provide an ~~optimal~~<u>optimized</u> sequence of FMA (fused multiply-

add), FMS (fused multiply-subtract), and/or FNMA (fused negate multiply-add)

instructions based on the correct sign combination and canonical form of the DAG as

compiled code for computing the incoming expression.

The following is an examiner's statement of reasons for allowance:

The prior art (e.g. "Automatic Generation of Implementations for DSP Transforms

on Fused Multiply-Add Architectures" by Voronenko) teaches generating optimized FMA

code using DAGs (pg. 2, col. 1 pars. 3-4 "DAGs. Our method for generating FMA

algorithms [operates on] an equivalent representation of the algorithm as a directed

acyclic graph or DAG describing the data flow of the computation") by calculating the

optimized FMA code (see e.g. Algorithm 1 bridging pp. 2 and 3).

The prior art taken alone or in combination does not teach or suggest generating

optimized FMA code by comparing FMA DAGs representing expressions in the code to

predetermined, optimal, FMA DAGs stored in a pattern table and generating the code

accordingly, as is claimed.

Any comments considered necessary by applicant must be submitted no later

than the payment of the issue fee and, to avoid processing delays, should preferably

accompany the issue fee. Such submissions should be clearly labeled "Comments on

Statement of Reasons for Allowance."

Conclusion

Any inquiry concerning this communication or earlier communications from the

examiner should be directed to JASON MITCHELL whose telephone number is

(571)272-3728. The examiner can normally be reached on Monday-Thursday and

alternate Fridays 7:30-5:00.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's

supervisor, Bullock Lewis can be reached on (571) 272-3759. The fax phone number

for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the

Patent Application Information Retrieval (PAIR) system. Status information for

published applications may be obtained from either Private PAIR or Public PAIR.

Status information for unpublished applications is available through Private PAIR only.

For more information about the PAIR system, see http://pair-direct.uspto.gov. Should

you have questions on access to the Private PAIR system, contact the Electronic

Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a

USPTO Customer Service Representative or access to the automated information

system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

/Jason Mitchell/

Primary Examiner, Art Unit 2193